중급 프로젝트 발표

프로젝트명: 살피다

코드잇 AI 4기 3팀

김명환, 신승목, 오형주, 이민규

[프로젝트저장소] [시연영상]

[helper-hwp]

목차

1.1 미션 배경 및 목표

1. 정부나라장터 환경

매일 수 백 건의 입찰 공고

제안 요청서(RFP): 수 백 페이지

→ 문서 조사와 분석 부담

2. 프로젝트 추진 배경

대량의 입찰 문서 분석

핵심 정보 추출

사용자의 질문에 답변

→ RAG 시스템

3. 핵심 해결 과제

(1) **HWP**, PDF 문서 처리

(2) 벡터 임베딩 기반 검색

(3) LLM 기반 응답 시스템

(4) 사용자 친화적 UI

4. 최종 목표

OpenAI를 활용하여 나라장터의 기본 서비스가능여부 검토, 구현하는 프로토타입 시스템 개발

1.2 프로젝트 정보

1. 프로젝트명 및 팀 구성

프로젝트명: 살피다

팀 구성 및 역할

신승목(데이터 엔지니어): 문서 전처리 및 DB 파이프라인

김명환(머신러닝 엔지니어): 임베딩 변환, FAISS 저장, 로직 검토

이민규(AI 리서쳐): LLM 기반 프롬프트 엔지니어링, RAG 평가

오형주(프론트엔드 엔지니어): UI 개발(Streamlit) 및 시스템 통합

2. 개발 기간 및 일정

개발 기간: 2025년 11월 10일 ~ 11월 28일
(총 3주)

1주차(11/10~11/14): 기반 구축 단계

2주차(11/17~11/21):
핵심 기능 개발 및 모듈 통합

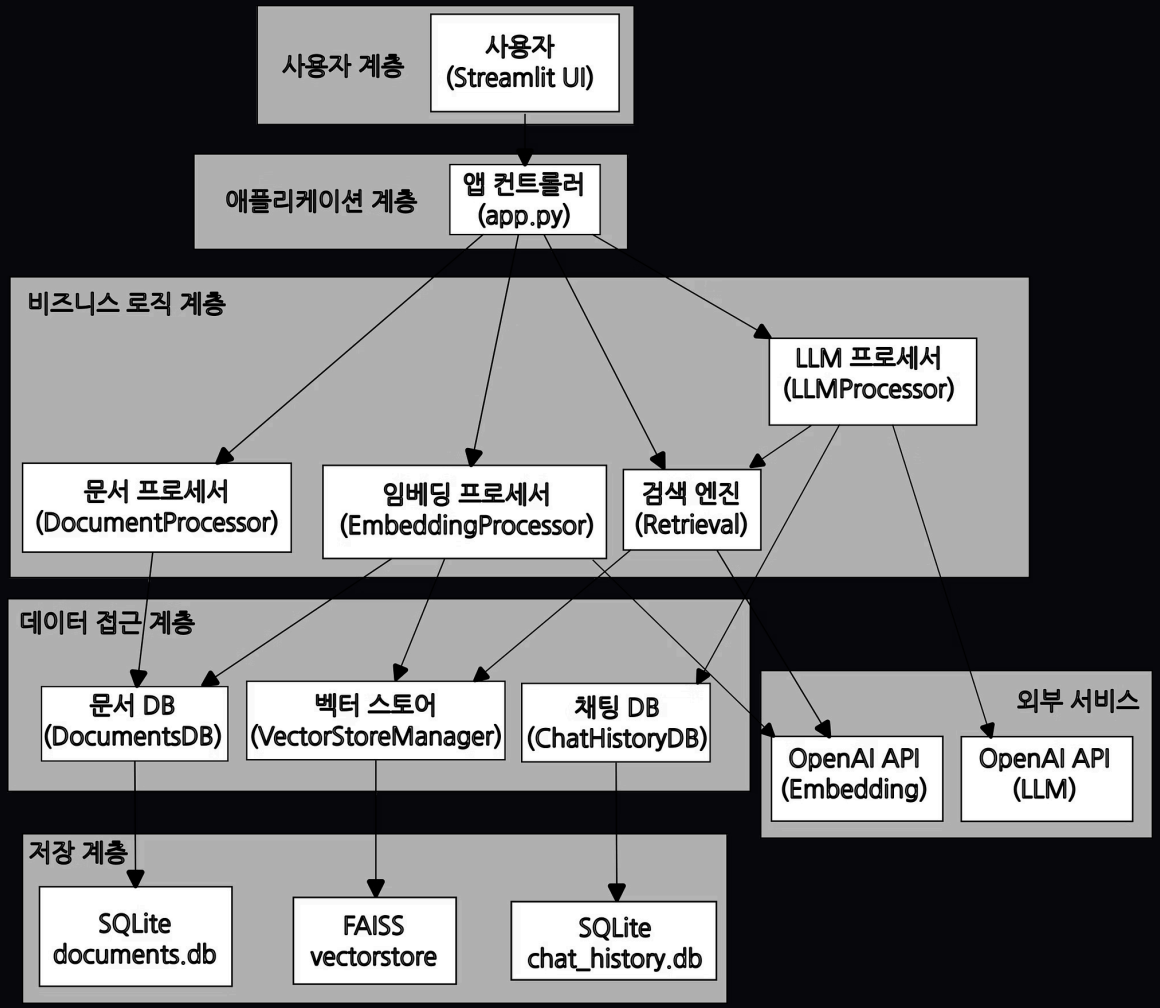
3주차(11/24~11/28): 최적화 및 마무리 단계

1.2 프로젝트 정보

3. 기술 스택

기능	라이브러리
문서 처리	PyMuPDF, pymupdf4llm, 자체 개발 **helper-hwp** 라이브러리
데이터 저장	SQLite (메타데이터, 채팅 이력)
임베딩/검색	OpenAI text-embedding-3-small 모델, FAISS, LangChain
LLM 계층	OpenAI GPT 시리즈 모델 (gpt-5-mini, gpt-4o), ConversationChain, ConversationSummaryMemory
사용자 인터페이스	Streamlit 프레임워크

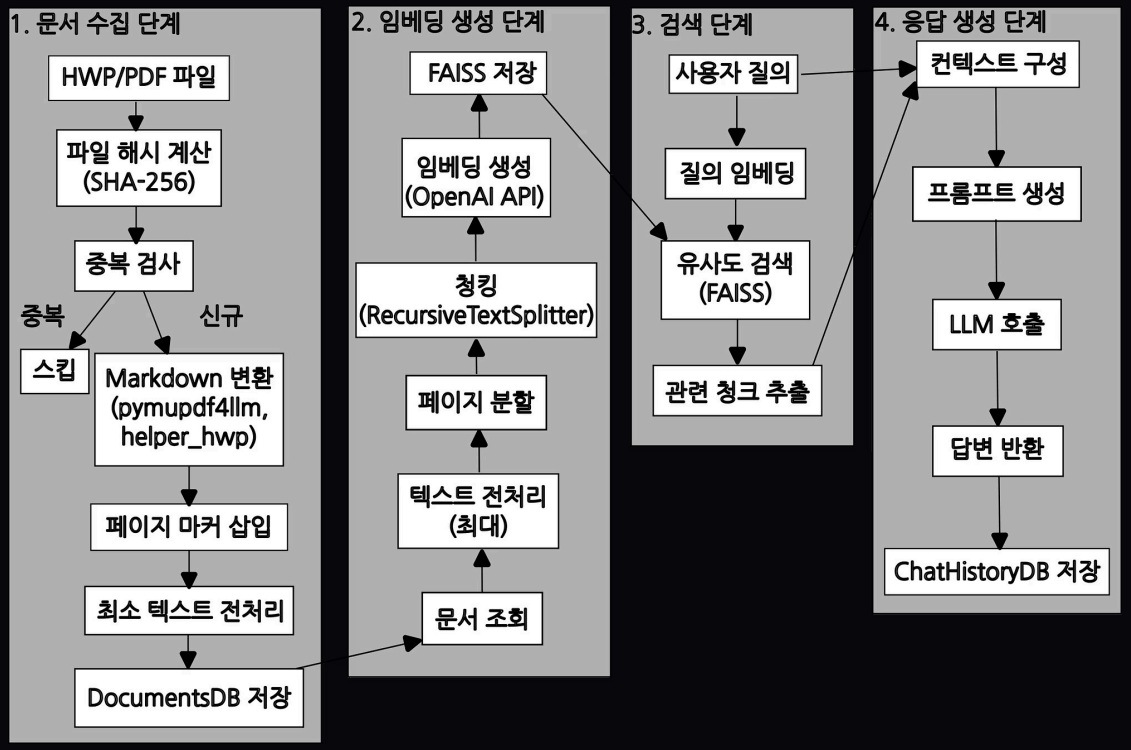
1.3 시스템 Architecture



1. 계층화 구조 채택

- 사용자 계층: RAG 시스템 이용
- 애플리케이션 계층:
  사용자 요청 → 비즈니스 로직
- 비즈니스 로직 계층: 핵심 로직 처리
- 데이터 접근 계층: DB, 벡터 인덱스 관리
- 저장 계층: SQLite, FAISS

1.3 시스템 Architecture



2. 데이터 파이프라인 흐름

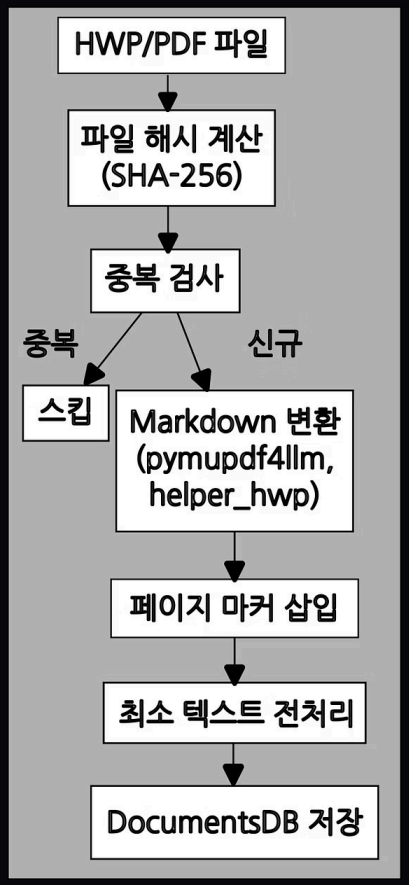
- 문서 수집: 파일 해시 기반 중복 검사, DB 저장
- 임베딩 생성: 청킹, 임베딩(1536차원) 후 FAISS 인덱스에 저장
- 검색: 사용자 질의 임베딩, 유사도 검색
- 응답 생성: LLM이 답변 생성,
  질의/답변 및 검색된 청크 → DB

1.3 시스템 Architecture

3. 주요 기술 구성 요소

모듈	역할	핵심 기술 및 특징
Config	중앙 집중식 설정 관리	config.json 파일, 싱글톤 패턴
DocumentProcessor	문서 변환	PyMuPDF, pymupdf4llm, **helper-hwp,** 페이지 마커 삽입
EmbeddingProcessor	청킹 및 임베딩 생성	3단계 전처리 파이프라인, RecursiveCharacterTextSplitter, **OpenAI text-embedding-3-small**
VectorStoreManager	벡터 인덱스 관리	**FAISS** (LangChain 래퍼), IndexFlatL2 기반 검색, **Document.metadata**에서 통합 관리
Retrieval	검색 수행	청크 기반 검색, 페이지 기반 검색 모드 지원, 메타데이터 필터링
LLMProcessor	응답 생성	OpenAI **GPT 모델**, LangChain ConversationChain 및 ConversationSummaryMemory로 컨텍스트 유지
ChatHistoryDB	대화 이력 관리	SQLite, 세션 기반 그룹화, 검색된 청크 정보(JSON) 저장
UI Module	사용자 인터페이스	**Streamlit** (세션 상태 관리: st.session_state), @st.cache_resource를 통한 DB 인스턴스 캐싱

2.1 문서 수집 전략



1. 문서 수집: 파일 해시 기반 중복 검사

- SHA-256 해시 이용
- 중복 검사: 파일 입력 시 해시 값 계산, 신규 파일만 처리
  DB에 이미 존재할 경우 중복으로 판단 → 변환 및 저장 생략

2. Metadata 추출 및 관리

- 파일 Metadata: 파일 해시, 파일명, 파일 크기, 총 페이지, 생성 시각, 수정 시각
- 페이지 수: (PDF)PyMuPDF, (HWP)Markdown 40줄마다 페이지 마커 추가
- 시각 정보: KST(ISO 8601 형식)

2.2 원본 전처리

1. Markdown 변환

PDF: pymupdf4llm

2초당 1페이지 처리

HWP: helper_hwp 개발

문서 1개당 1초 이내 처리

2. 최소 전처리만 수행

원본 텍스트 내용 및 구조 보존

(1) 공백 및 탭 정규화
(2) 3회 이상 개행을 2회로 축소
(3) 각 라인 앞뒤 공백 제거

3. 페이지 마커 삽입

청킹 시 페이지 단위 분할

오류 발생 시 특수 마커

검색 결과 출처 표시 → UX 향상

2.3 데이터베이스 설계

1. documents_db 스키마

파일 해시, 파일 이름,
총 페이지, 파일 크기

Markdown 내용 (분할 저장)
분할 저장 인덱스(0~4)
생성 시각, 최종 수정 시각

2. 파일 메타데이터 관리

Create, Read, Update, Delete
(= CRUD)

테이블에 정보 추가
파일 해시 조회 (중복 검사)
파일 통계 정보 반환

3. 재현성 보장 메커니즘

파일 해시로 동일 문서 1회만 저장

Markdown 원본: 다양한 임베딩 실험

타임스탬프 추적

2.4 중복 데이터 분석 및 품질 개선 효과

1. 해시 기반 중복 검출

해시값: 20cd…0a7
(저장) BioIN 의료 기기산업…
(스킵) 한국보건산업진흥원…

해시값: fe07…3e8
(저장) 국가과학기술지식정보…
(스킵) 한국한의학연구원…

2. 검출 과정

DB에서 저장 개수 확인
→ 출력 로그 검토
→ DB에 해시 검색
→ (원본) 동일 확인

3. 저장 공간 절약

원본 파일 기준 1%
(1.6 MB/162 MB)

DB 파일 17.0 MB
(원본 데이터 중 98개 저장)

```
2025-11-26 17:09:10 [I] __main__ - 100개 중 75번째 파일 처리: **한국보건산업진흥원_의료기기산업 종합정보시스템(정보관리기관) 기능.hwp**
2025-11-26 17:09:10 [I] [DOCP] - HWP 처리 시작: 한국보건산업진흥원_의료기기산업 종합정보시스템(정보관리기관) 기능.hwp
2025-11-26 17:09:10 [D] [DOCDB] - 검색 완료: 1건 (hash 모드, 검색어: 20cdb1e78194ab2496ca941e3c9a4a2b74558dd007535964517584cab67be0a7)
2025-11-26 17:09:10 [I] [DOCP] - 이미 처리된 파일 (skip): 한국보건산업진흥원_의료기기산업 종합정보시스템(정보관리기관) 기능.hwp
2025-11-26 17:09:35 [I] __main__ - 100개 중 98번째 파일 처리: **한국한의학연구원_통합정보시스템 고도화 용역.hwp**
2025-11-26 17:09:35 [I] [DOCP] - HWP 처리 시작: 한국한의학연구원_통합정보시스템 고도화 용역.hwp
2025-11-26 17:09:35 [D] [DOCDB] - 검색 완료: 1건 (hash 모드, 검색어: fe07779f264abfd5f420f332adb65779a33142a5506581345f5cbe4803be53e8)
2025-11-26 17:09:35 [I] [DOCP] - 이미 처리된 파일 (skip): 한국한의학연구원_통합정보시스템 고도화 용역.hwp
```

2.5 나라장터 공고 → DB

1. 입찰 공고 정보 조회

API에서 Service Key, 시작일, 종료일 입력 (10초)

→ 조달청_나라장터 입찰공고 서비스 URL 생성

http://apis.data.go.kr/1230000/ad/BidPublicInfoService/getBidPblancListInfoCnstwk(공사 관련 공고)

2. API 응답 중 첨부문서 → DB

(1) 임시 디렉토리 생성 및 다운로드 (30초 timeout)

(2) 예외 처리: Long Path Prefix → 절대 경로로 변환

(3) 다운 완료 시: HWP, PDF → Markdown → DB

(4) DB 저장 완료 시 임시 파일 제거

3.1 임베딩 전처리 전략

1. 3단계 전처리 파이프라인

단계	수행 시점	주체 (메서드)	목표	주요 작업	저장 위치
1차: 최소	HWP, PDF 변환 직후	DocumentProcessor.clean_markdown_text	원본 보존	형식 문제 해결 (공백, 개행 축소)	DocumentsDB.text_content
2차: 최대	청킹 직전	EmbeddingProcessor.clean_markdown_text	임베딩 품질 최적화	마크업, 노이즈 제거 보호 블록 마스킹/복원	임베딩 입력
3차: 페이지별	페이지 단위 분할 후	EmbeddingProcessor.clean_page_text	최종 정제	페이지 마커 제거 최종 공백 정리	청크 텍스트

3단계 파이프라인의 장점: (1) DB 원본 보존한 채 다양한 전처리 실험 가능 (2) 페이지 마커로 정확한 페이지 분할 가능

3.1 임베딩 전처리 전략

2. 보호 블록 마스킹 기법

코드 예제, 수학 공식, 다이어그램 등 중요한 정보 보호

(1) 식별 및 치환

코드 블록(```) → XPROTECTEDXCODE3XnX
수식 블록($$, $) → XPROTECTEDXMATHXnX
페이지 마커(PAGE_MARKER) → XPROTECTEDXMARKERXnX

(2) 전처리 수행: 마스킹 후 전처리 작업에 영향 X

(3) 복원: 전처리 완료 후 원본으로 치환

3. Markdown 요소 제거 및 정제

보호 블록 마스킹 후, 형식을 위한 마크업 제거

제거:

HTML 태그, 이미지, 링크, 강조 기호,
헤더, 인용구, 리스트마커

탈출문자 처리, 공백 정리

효과: 임베딩 모델이 순수한 텍스트의 의미에 집중

3.2 청킹 전략

1. RecursiveCharacterTextSplitter

Separators: 문단 > 문장 > 단어 > 문자 순 분할

chunk_size: 기본값 1500, token_count 기준

chunk_overlap: 기본값 300

length_function: tiktoken 기반 커스텀 함수 사용

2. 페이지 단위 분할 방식

(1) 페이지 마커 기준으로 분할

(2) chunk_size보다 작은 단일 페이지는 버퍼 사용: 인접 페이지 텍스트 누적, 시작/종료 페이지 기록

(3) 청킹 트리거: 버퍼 크기 도달 또는 마지막 페이지

(4) 각 청크에 페이지 정보 저장, 사용자에게 출처 명시

3.2 청킹 전략

3. 메타데이터 보존 전략

메타데이터 필드	설명	활용 목적
file_hash	원본 파일의 SHA-256 해시값	특정 파일의 청크 식별, 파일 단위 벡터 삭제
file_name	사용자 친숙한 파일 이름	검색 결과 표시
start_page/end_page	청크가 속한 페이지 범위	원본 문서의 정확한 위치 추적
chunk_type	청크 생성 방식 (single, merged, split)	청킹 전략 분석 및 개선
chunk_index	동일 파일 내 청크의 순서 인덱스	청크 정렬, 인접 청크 검색
embedding_config_hash	파일 해시와 임베딩 설정의 통합 해시값	재현성 보장 및 재임베딩 필요성 자동 판단
chunk_hash	청크 텍스트 내용의 SHA-256 해시값	증분 업데이트 시 내용 변경 감지
config_*	청크 생성 당시의 설정값들	나중에 동일한 조건으로 청크 재생성
embedding_version	사용된 임베딩 모델 이름	모델 변경 시 추적 및 마이그레이션
created_at	청크 생성 시각 (ISO 8601)	시간 기반 필터링 및 데이터 갱신 분석

JSON 직렬화 가능한 형태로 메타데이터 저장, LangChain의 Document 객체에 포함, VectorStoreManager는 FAISS 인덱스와 함께 저장

3.3 벡터 임베딩

1. OpenAI text-embedding-3-small

특징	설명
벡터 차원	1536 차원
비용 효율성	백만 토큰당 0.02 달러 (이전 모델 대비 5배 저렴)
처리 속도	빠른 API 응답 시간, 배치 처리 지원
다국어 지원	한국어 포함 100개 이상
컨텍스트 길이	최대 8191 토큰 처리 가능
배치 처리	기본값 100, 효율성 증가

2. 임베딩 설정 해시

목적: 재현성 보장, 설정 변경 추적

정보: 파일 해시와 임베딩 관련 모든 설정 결합하여 계산한 SHA-256 해시값

embedding_config_hash =
SHA-256(Sort(file_hash, chunk_size, chunk_overlap, embedding_model, …))

특징: 벡터 생성에 관련된 모든 요소 포함, 설정이 달라지면 재임베딩 필요성 자동 판단

활용: 중복 방지, 재임베딩, 실험 추적 등

3.3 벡터 임베딩

3. 재현성 보장 메커니즘

재현성 수준	핵심 메커니즘	보장 내용
파일 수준	file_hash	동일한 PDF 파일 -> 동일한 해시값 및 text_content
설정 수준	embedding_config_hash	동일한 설정 -> 동일한 청크 생성 (Config 파일로 설정 복원 가능)
청크 수준	chunk_hash	동일한 텍스트 -> 동일한 임베딩 벡터 (OpenAI 모델은 결정론적)
메타데이터 수준	config_* 필드	청크 생성 당시의 설정을 완전히 기록
버전 관리	embedding_version	임베딩 모델 업데이트 시 재임베딩 필요성 판단

3.4 FAISS 벡터 저장소

1. 인덱스 구조 및 관리

인덱스 타입: IndexFlatL2
→ 검색 결과 정확도 보장

docstore(Document 저장),
index_to_docstore_id(ID 매핑)
→ 벡터와 원본 문서 연결

save_local, load_local
→ FAISS 인덱스, 메타데이터
저장 및 복원

2. 메타데이터 매핑 전략

FAISS 자체는 메타데이터 지원 X
→ 추가 자료 구조 chunk_map

chunk_map = Dict[
(file_hash, chunk_index),
(faiss_idx, chunk_hash,
embedding_config_hash)]

chunk_map 활용 목적:

빠른 중복 검사, 내용 변경 감지,
설정 변경 감지, 효율적인 삭제

3. 검색 성능 최적화

top_k 조정: 기본값 5로 검색 시간 최소화

메타데이터 필터링 지원

인덱스 재구성 최소화,
대량 업데이트 시 재구성

3.5 다단계 검색 시스템 (Multi-Stage Retrieval)

1. 1차 광범위 검색 (Broad)

목표: 전체 문서에서 관련 청크 식별,
→ 재현율(Recall) 향상

(1) 질의 임베딩:
OpenAIEmbeddings, 1536차원

(2) FAISS 검색:
L2 거리 기반 검색,
top_k(=5) 청크, 거리점수

(3) 결과 반환:
청크의 텍스트, 메타데이터(file_hash, start_page, distance 등)

2. 2차 심층 검색 (Deep)

목표: 1차 검색 문서에서만 검색 수행,
→ 정밀도(Precision) 향상

(1) 메타데이터 필터링:
특정 file_hash 가진 청크만 검색

(2) 검색 후 처리:
1차 결과에 필터 적용 → 최종 결과 선택

(3) 효과:
정확한 답변 얻을 확률 증가,
불필요한 노이즈 제거

3. 파일 해시 기반 필터링

2차 심층 검색의 핵심 메커니즘

(1) 정밀도 향상:
다른 문서의 유사 표현(FP) 차단

(2) 컨텍스트 일관성:
특정 문서에 대화 집중 → 답변 품질 향상

(3) 사용자 경험:
하나의 문서를 깊이 탐색하는 경험 제공

3.6 전처리 효과 분석

1. 토큰 절약 효과 (21.5%)

원본 36,146자 → 전처리 후 28,359자

(1) 공백/개행 정리:
15.4% 절약

(2) Markdown 요소 제거:
23.3% 추가 절약(HTML 태그 등)

(3) 빈 테이블 행 제거:
47.4% 극적인 효과

2. 비용 절감 분석

text_embedding-3-small 모델 비용:
백만 토큰당 0.02 달러

문서 당 9000 토큰 → 7000 토큰
연간 약 15.3 달러 (8만건 가정)

LLM 응답 생성 비용도 간접 감소

3. 검색 품질 개선

(1) 임베딩 모델이 내용에만 집중
→ 정확한 의미 표현

(2) 동일 청크에 더 많은 내용
→ 정보 밀도와 검색 확률 증가

(3) 다양한 문서에 통일된 변환
→ 임베딩 공간 의미적 유사성 향상

(4) 형식적 유사성으로 인한 잘못된 매칭
(False Positive) 감소

4.1 LLM 프로세서 설계

1. ChatOpenAI 모델

모델 및 설정: gpt-5-mini(2025),
Temperature 기본값 0.0
→ 생성된 텍스트의 창의성과 일관성을 조절

모델별 특수 처리: gpt-5-mini는 temperature 1.0 자동 조정 (오류 방지)

API 키 관리(보안 및 편의성):
메서드 파라미터 > 환경변수 > Config

API 호출 추상화(LangChain):
재시도 로직(최대 3회), 스트리밍 응답,
토큰 카운팅 기능 자동 처리

2. 파라미터 최적화

Temperature: 0.0 (결정론적 답변)
→ 문서 기반 답변

Max_tokens: 50000 (응답 시간, 비용)
→ 대부분의 질문에 충분한 답변을 할 수 있는 값

Presence_penalty, Frequency_penalty (반복 억제): 0.0(기본값)
→ 문서 내용을 충실히 반영하기 위해 적용 x
→ 지나친 반복 답변시 조절 가능

모델별 파라미터 처리: generate_response 메서드로 모델별 자동 선택

4.2 프롬프트 엔지니어링

1. RAG 프롬프트 템플릿

프롬프트: LLM에게 작업 지시 (핵심 요소)
RAG 시스템 구조: 검색된 문서를 컨텍스트로 제공 → 답변

기본 프롬프트 구조:
(1) 시스템 역할 정의 지시문
(2) 컨텍스트: 검색된 청크들 + 출처 정보
(3) 질문: 사용자의 원래 질의
(4) 답변 유도: 라벨(ex. "답변:") 사용하여 즉시 답변 유도

템플릿 관리

- 여러 프롬프트 템플릿을 관리
- PromptLoader: 외부에서 프롬프트 템플릿 로드, JSON 저장
- 프롬프트 코딩 우선순위: 항상 유효한 프롬프트 사용 보장
  key값 순서대로 프롬프트 존재여부를 파악하여 불러옴

4.2 프롬프트 엔지니어링

2. 컨텍스트 구성 전략

구분	청크 기반	페이지 기반
사용 메서드	Retrieval.search	Retrieval.search_page
구성 단위	각 청크 독립적, 순서대로 나열	페이지 단위 병합 텍스트 (더 넓은 문맥)
출처 정보	문서 번호, 파일명	파일명, 페이지, 유사도(4자리)
포맷	이중 개행으로 청크 분리	(예: "출처 1: 공고문.pdf, 페이지 5, 유사도=0.1234")

3. 한국어 프롬프트

(1) 프롬프트 언어 : 구성 요소를 모두 한국어로 작성 → 한국어로 작동하도록 유도

(2) 경어 수준 조정: 명시적으로 존댓말 사용 지시

(3) 용어 일관성: "입찰 및 조달 분야의 전문 용어를 정확히 사용하세요" → 제안요청서, 과업지시서 등의 도메인 특화 용어 오용 감소

(4) 문장 구조: 간결하고 명확한 문장 요청 → 답변 구조 개선

(5) 출처 표시 요구

4.3 대화 이력 관리

1. ChatHistoryDB 스키마

SQLite에 채팅 세션과 메시지 저장

테이블 구조

(1) chat_sessions 테이블 (대화 세션 메타데이터)

(2) chat_messages 테이블 (실제 대화 내용)

데이터베이스 파일:

data/chat_history.db

데이터 무결성 및 성능:

외래 키 제약으로 무결성 보장, 인덱스 자동 생성으로 빠른 조회 속도

2. 세션 기반 대화 관리

세션 생성: 고유 ID 자동 생성

메시지 추가: 검색된 청크 JSON 문자열로 저장

세션/메시지 조회: 모든/특정 세션의 메시지 반환

세션 삭제: 관련 메시지 모두 삭제 (CASCADE 옵션)

통계 정보: 사용 현황(총 세션 수, 메시지 수)

3. 검색 청크 메타데이터 저장

대화 이력에 검색 청크 정보 함께 저장
→ 답변 근거 추적, 검색 품질 평가, 디버깅 및 개선

청크 메타데이터: 파일해시, 파일명, 페이지 번호, 거리 등

JSON 지원하는 문자열로 변환

기대 효과:
(1) 사용자에게 원본 문서 위치 안내
(2) 검색 품질 평가, 임계값 조정에 활용

4.4 메모리 및 요약

1. 대화이력 요약 관리

긴 대화 → 토큰 한도 및 응답 시간 문제
ConversationSummaryMemory 활용

가장 오래된 대화를 요약으로 대체,
새로운 메시지 유지 → 최근 컨텍스트 보존

효과: 비용 절감, 응답 속도 유지,
컨텍스트 연속성 유지

2. 한국어 요약 프롬프트

한국어 대화 최적화된 요약 생성

PromptTemplate 객체로 정의,
입력 변수: 기존 요약과 새로운 대화 내용

템플릿 구조: 기존 대화와 새 대화의 라벨 제시 후 지시문으로 누적 요약 지시

4.5 RAG 평가 시스템

1. LLM Judge 기반 평가

평가 모델: gpt-5

프롬프트: "당신은 RAG 시스템의 품질을 평가하는 최고 전문가 LLM Judge 입니다"

평가 데이터 구조: 질의, 문서, 질의 답변 구분

3. JSON 형식 결과 반환

응답: 원래 질의, 생성 답변, 4개 지표, 평가

응답을 정규표현식 처리 후 parsing

결과를 DB에 저장하여 품질 추적, 통계 분석

2. 4대 평가 지표

지표명	한글 용어	평가 내용	핵심 목적/의미
Faithfulness	충실성/근거성	생성된 답변이 컨텍스트에 충실한가	환각 감지 및 시스템 신뢰성 (5점: 모든 정보를 문서에서 확인 가능)
Context Relevance	문맥 관련성	검색된 문서와 질의의 연관성	검색 단계의 품질 측정 (낮다면 검색 전략 개선 필요)
Answer Accuracy	답변 정확도	질의에 대한 답변의 정확성	전체 시스템이 질문에 올바르게 답하도록 잘 구성되어 있는지
Answer Relevance	답변 관련성	답변과 질의의 직접적 연관성	프롬프트 설계 효과 측정

5.1. Streamlit 기반 UI 설계

1. 페이지 구조 및 레이아웃

단일 페이지, 사이드바와 메인 영역

사이드바: 주요 설정 및 제어 기능

메인 영역: 채팅 인터페이스
사용자 메시지 + 어시스턴트 응답 (시간순)

입력 영역: 화면 하단의 고정 컴포넌트

반응형 디자인: 모바일 환경 사용 가능

2. 채팅 인터페이스 구현

대화 이력 저장 및 유지

스트리밍 응답 → 사용자 대기시간 단축

spinner: 검색 및 응답 생성 중임을 알림

에러 처리: 사용자 친화적인 메시지,
기술적 세부사항 은폐, 구체적 해결법 안내

새로운 메시지 추가 시 자동 스크롤로 확인

메시지 출처 확인 가능 → 답변 신뢰성 향상

3. 세션 관리 UI

사이드바에서 사용자가 여러 대화 관리

새 세션 생성: 초기화 + 새로운 대화 시작

세션 목록: 각 세션 이름, 최근 활동 시각 등

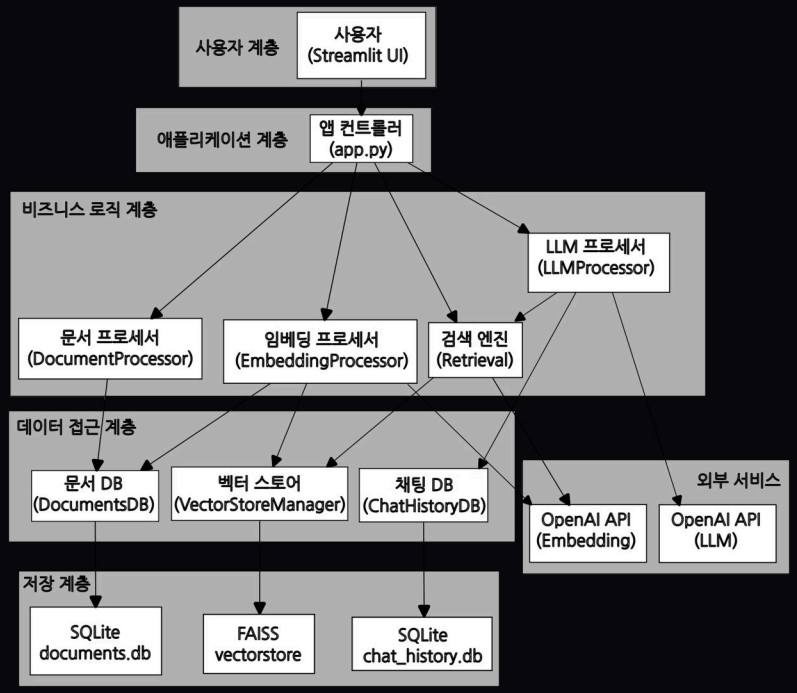
세션 전환: ChatHistoryDB에서 로드,
현재 세션 상태는 자동 저장

세션 삭제: 확인 절차 추가하여 실수 방지

활성 세션: 색상과 아이콘으로 구분

세션 통계: 사이드바 하단 (대화, 메시지 수)

5.2. 시스템 통합 아키텍처



1. 모듈 간 데이터 흐름

(1) 사용자 질문 (Streamlit UI 입력) → app.py 이벤트 핸들러 수신

(2) Retrieval 모듈: 질의 텍스트 전달하여 관련 문서 검색

(3) Retrieval → VectorStoreManager: FAISS 인덱스에서 유사도 검색 실행, 텍스트 및 메타 데이터 반환

(4) LLMProcessor 호출: 검색 chunk + 사용자 질의 → context → LLM

(5) LLMProcessor → OpenAI API: 응답 생성

(6) LLMProcessor → ChatHistoryDB: 질의와 응답(검색 청크 포함) 기록

(7) app.py로 답변 반환 → Streamlit UI를 통해 사용자에게 표시

에러 전파: 하위 모듈의 예외가 로깅 후 상위 모듈로 전파,
app.py에서 사용자 친화적인 오류 메시지로 변환

5.2. 시스템 통합 아키텍처

2. Config 기반 중앙 집중식 설정 관리

Config 클래스: Singleton Pattern, 애플리케이션 전체에서 동일한 인스턴스 공유, 일관된 설정과 메모리 효율성 보장

설정 파일: JSON 형식으로 가독성과 편집 용이성, 설정 누락 시 클래스 내부에 정의한 기본값 사용

동적 설정 변경: top_k 설정은 UI를 통해 런타임에 동적 조정 가능, 즉시 다음 질의부터 적용

환경 변수 override: 민감 정보는 환경 변수 우선 확인하여 보안 강화 및 배포 환경의 유연성 확보

설정 검증: app 시작 시 필수 설정 및 유효성 검증하여 런타임 에러를 사전에 방지

3. 로깅 및 모니터링

포괄적인 로깅 체계 → 시스템 동작 추적 및 문제 진단 용이

로그 레벨: DEBUG, INFO, WARNING, ERROR, CRITICAL 5단계 중요도

일관된 로그 포맷: 타임스탬프(KST), 로그레벨, 모듈명, 함수명, 메시지

파일 로깅: 로그 파일이 일정 크기 초과 시 날짜별로 자동 저장

콘솔 로깅: 개발 환경에서 실시간 피드백 제공, 시각적으로 레벨 구분

성능 로깅: 검색, 임베딩, LLM 응답 생성 등 작업 실행 시간 측정 및 기록하여 병목 지점 식별

에러 로깅: 전체 예외 정보를 상세히 기록

5.3. 추가 기능

1. Top-K 파라미터 조정 (1 ~ 20)

(1) UI 설정:
슬라이더를 활용하여 범위 설정, 기본값: 5

(2) 안내문 출력:
값이 높을수록 많은 문서 / 시간 및 비용 증가

(3) 미리보기:
슬라이더 조정 시 마지막 질의 재검색,
청크의 제목이나 첫 문장 미리보기

(4) 동적 조정 제안:
질의의 복합도 → Top-K 추천

2. 문서 업로드 및 인덱싱

사용자가 직접 문서 추가 및 분석 가능

(1) 파일 업로드: HWP, PDF 가능

(2) 중복 검사:
파일 해시 계산 → 기존 DB와 비교 중복 시 처리 중단

(3) 문서 처리: 진행 상황 표시
업로드 → 파싱 , 마크다운, 최소 전처리 → DB 저장 → 임베딩 생성

(4) 임베딩 생성 선택: 즉시 / 나중에 수행 선택

(5) 오류 처리: 오류 메시지 표시, 데이터 롤백 → DB 일관성

5.3. 추가 기능

3. 공공데이터 포털 업데이트

공공데이터 포털에서 특정 기간 정보 수집

(1) 시작시각/종료시각: UI 입력 → DocumentDB 업데이트

(2) 중복 검사: 파일 해시 기반 중복 시 업로드 중단

(3) 문서 처리: 미중복 시 처리, 진행상황 표시

(4) 오류 처리: 오류 메시지 표시, 데이터 롤백 → DB 일관성

4. 검색 결과 시각화

(1) 청크 카드: 각 검색 결과를 접거나 펼칠 수 있음
헤더: 문서명, 페이지 번호, 유사도 점수
펼칠 시 청크 텍스트, 질의와 일치하면 강조

(2) 유사도 점수 시각화:
유사도 0~1 정규화, 색상막대 표현 → 직관성

(3) 문서 분포 차트: 파이 / 막대 차트
출처 문서 시각화, 특정 문서 필터링

6.1. 프로젝트 성과

1. 기술적 성과

(1) 시스템 아키텍처: 계층화 설계
→ 모듈 독립성과 유지보수성 확보

(2) 문서 파이프라인 완전 자동화
SHA-256 해시 기반 중복 검출:
저장 공간과 임베딩 비용 절감

(3) 3단계 전처리 과정: 효율 증가
→ 다양한 임베딩 실험 가능
→ 21.5% 토큰 절약

(4) 임베딩 및 벡터 저장소 확장성
OpenAI embedding-3-small, 1536
FAISS IndexFlatL2 인덱스
포괄적인 메타데이터 첨부
→ 출처 추적 및 재현성 보장

(5) 다단계 검색 시스템 : PR 균형
1차 광범위 검색: 전체 DB 유사도
2차 심층 검색: 파일 해시, 문서 고정

(6) LLM 통합
gpt-5-mini: 빠른 속도, 낮은 비용
긴 대화도 토큰 한도에서 컨텍스트 유지

(7) LLM Judge 기반 평가 시스템
Faithfulness, Context Relevance,
Answer Accuracy, Answer Relevance

(8) UI/UX:
Streamlit 기반 : 직관성과 기능성
세션 관리, 스트리밍 응답, 검색 시각화

6.1. 프로젝트 성과

2. 정량적 지표

구분	지표 내용	정량적 성과
전처리 효율성	원본 대비 평균 토큰 절약률	21.5 %
HWP 전처리 속도	문서 1건 당 처리 속도	문서 1건당 1초 (PDF: 2초당 1페이지)
비용 절감 효과	문서당 임베딩 비용 절감 (USD)	0.000193 (문서 8만개 기준 $15.3)
중복 제거 성과	감지 및 제거된 중복 문서 수	2 건
응답 생성 시간	gpt-5-mini 평균 응답 시간	2 초 ~ 5 초
DB 응답 시간	ChatHistoryDB 메시지 조회 시간	10ms 이내
FAISS 검색 시간	수천 벡터 대상 최근접 이웃 검색	10ms ~ 50 ms

6.2. 핵심 인사이트

1. RAG 시스템 설계 원칙

전처리: 노이즈 제거 + 의미 보존

청킹: 페이지 경계 또는 의미 단위

다단계 검색: 후보 선정 후 정밀 검색

Metadata: 해시 기반 재현성

한국어는 영어에 비해 청크 크기 20% 크게

2. 팀 협업 및 프로젝트 관리

인터페이스 우선 설계:
각 모듈의 인터페이스 문서화
→ 병렬 개발과 원활한 통합 가능

점진적 통합으로 위험 분산:
매주마다 완성된 모듈을 통합 및
End-to-end test 수행

3. 기술 선택 Trade-off

OpenAI 모델 의존성: 비용증가
→ 향후 오픈소스 모델 전환 가능성

SQLite: 초기 개발과 배포에 적합,
PostgreSQL (대규모)

FAISS IndexFlatL2: 소규모에 적합,
IndexIVFFlat, IndexHNSW (대규모)

6.3. 한계점 및 개선 과제

1. 현재 시스템의 제약

단순 유사도 기반 검색 → 의도 분류, 엔티티 인식 추가

영어 중심 모델 → 한국어 특화 모델 파인튜닝

소규모 데이터용 시스템 → 대규모: PostgreSQL, IndexHNSW

단일 사용자 환경 → 실제 배포 시 보안 필요

2. 사용자 피드백 부재

팀 내부 테스트에만 의존: 파일럿 사용자 피드백

조달 분야 전문가나 법률 자문 등 전문적 검증 필요:
전문성 확보, 출력의 검토 및 개선

6.4. 향후 개발 계획

1. 단기 계획 (1개월 이내)

(1) 파일럿 사용자 테스트:
사용성, 정확성, 유용성 피드백

(2) 검색 품질 개선:
하이브리드 (BM25 + 벡터 검색)
Reranking → 검색 정밀도

(3) 프롬프트 최적화:
Few-shot 예제, CoT 프롬프팅

(4) 질문과 응답 캐싱:
LLM 호출 중복 방지 및 비용 절감

2. 중기 계획 (3개월 이내)

(1) 한국어 특화 모델 파인튜닝:
입찰 공고 도메인 한국어 Corpus 활용,
LLaMA, Mistral, Solar 등 오픈소스

(2) DB migration: PostgreSQL (대규모)

(3) FAISS 인덱스 업그레이드:
IndexIVFFlat, IndexHNSW (대규모)

(4) 멀티모달 지원:
OCR 및 표 인식 → 이미지와 표 처리

(5) 보안 기능 구현:
OAuth 2.0 기반 인증, API 키 보호 등

3. 장기 계획 (6개월 이상)

(1) 자동 문서 업데이트 시스템:
나라장터 연동, 자동 수집 및 업데이트

(2) 대화형 분석 도구 개발:
자연어 질의를 SQL 또는 Python 코드 변환, 데이터 탐색 및 시각화 기능 제공

(3) 사용자의 질의 및 관심 분야 학습
→ 새로운 공고를 능동적으로 추천

(4) 엔터프라이즈 기능:
팀 단위 협업, Workflow 자동화, 대시보드 제공 등 조직 차원의 사용 지원

6.5. 감사의 말

본 프로젝트는 코드잇 AI 4기 교육 과정의 일환으로 수행되어, 교육팀의 체계적인 커리큘럼, 멘토님들의 기술적 조언, 그리고 팀원들의 전문성과 헌신 덕분에 성공적으로 마무리될 수 있었습니다.

팀원들의 헌신:
신승목님 (데이터 엔지니어): 데이터 파이프라인 및 DB 구축
김명환님 (머신러닝 엔지니어): 임베딩 처리, FAISS 인덱스 최적화, 다단계 검색 시스템 완성
이민규님 (AI 리서처): LLM 통합, 프롬프트 엔지니어링, 평가 시스템 구축
오형주님 (프론트엔드 엔지니어): Streamlit 기반 UI 개발 및 모든 모듈 통합

오픈소스 커뮤니티와 도구를 제공한 개발자들에게 깊은 감사를 표하며, 본 프로젝트가 RAG 시스템 구축을 고민하는 이들에게 기여하고 본 프로젝트가 RAG 시스템 구축을 고민하는 이들에게 기여하고 AI 기술 민주화에 보탬이 되기를 바랍니다.